```
/*
QSortProject
Definizione di 4 interfacce.
*/

import java.io.*;
import java.util.*;

interface sortable{//per sole strutture modificabili
    void sort();
}
interface ordered{//per soli preordini (riflessivi, transitivi) totali
    boolean ord(ordered s);
}
interface showable{
    String show();
}
interface readable<T>{
    T read();
}
```

```
/*
 QSortProject
 ADT per Sequenze Modificabili, a doppio scorrimento (Doubly-Linked Lists)
 definizione di una collezione astratta modificabile
 --- Esibita solo interfaccia.
 --- Implementazione "nascosta" (solo intestazione classe di implementazione)

*/
import java.io.*;
import java.util.*;


interface MutSeqADT<T>{
    MutSeqADT<T> pred();
    MutSeqADT<T> succ();
    T val();
    void valUpd(T v);
}


public class MutSeq<T> implements MutSeqADT<T>{
/* ----------- Stato oggetti intMutSeq ------------------------ */
    private MutSeq<T> Pred;
    private MutSeq<T> Succ;
    private T Val;
/*------------ Costruttori ----------------------------------- */
    public MutSeq(MutSeq<T> p, T v){//alias di add
        /* inserisce dopo p */
        Pred = p;
        Val = v;
        if (p!=null){
            Succ = p.Succ;
            p.Succ = this;
            if (Succ!=null) Succ.Pred = this;
        }
    }
```

```
/* ------------ Metodi per OPERAZIONI -------------------------- */
/* ------------ OSSERVATORI PER ACCESSO COMPONENTI ------------- */

    public MutSeq<T> pred(){
        return Pred;
    }
    public MutSeq<T> succ(){
        return Succ;
    }
    public T val(){
        return Val;
    }
/* ------------ MODIDICATORI COMPONENTI ------------------------ */

    public void valUpd(T v){
        Val = v;
    }
}







/*
 Estende MutSeq<T> in una classe sortable
 Avremmo potuto estenderla anche in showable e readable se T lo ammette!!.
 -- Esibita solo struttura dell'estensione.
 -- Implementazione "nascota" (avremmo potuto usare classi astratte)
*/

import java.io.*;
import java.util.*;


class IllegalArgsException extends Exception{
    public IllegalArgsException(String s){super(s);}
}


public class myMutSeq<T extends ordered> extends MutSeq<T> implements sortable{
/* ------------ Stato solo eredtato------------------------------ */
/*------------ Costruttori ------------------------------------- */

    public myMutSeq(myMutSeq<T> p, T v){
        super(p,v);
    }
/* overriding di pred e succ ---------------------------------- */

    public myMutSeq<T> pred(){
        return (myMutSeq<T>) super.pred();
    }
    public myMutSeq<T> succ(){
        return (myMutSeq<T>) super.succ();
    }
/* altri metodi osservatori ----------------------------------- */

    public myMutSeq<T> first(){/* leftmost element */
        if (pred()==null) return this;
        return pred().first();
    }
```

```
   public myMutSeq<T> last(){/* rightmost element */
       if (succ()==null) return this;
       return succ().last();
   }
/* ------------ MODIFICATORI COMPONENTI ------------------------ */

   public void swap(myMutSeq<T> q)throws IllegalArgsException{
       /* element swapping di this.val con q.val */
       if (q==null) throw new IllegalArgsException("swap");
       {
           T temp = val();
           valUpd(q.val());
           q.valUpd(temp);
       }
   }
/* implement sortable */
   public void sort(){
       class sortStuff{
           class pair{
               myMutSeq<T> left; myMutSeq<T> right;
               pair(myMutSeq<T> x,myMutSeq<T> y){
                   left = x; right = y;
               }
           }
           T separatore(myMutSeq<T> u) throws IllegalArgsException{
               if (u==null) throw new IllegalArgsException("separatore");
               return u.val();
           }
           pair divisione(myMutSeq<T> inf, myMutSeq<T> sup, T a){
               boolean met = false;
               myMutSeq<T> inf2 = inf;
               myMutSeq<T> sup1 = sup;
               met = met || (sup1 == inf2);
               while (!met) {
                   T x = inf2.val();
                   while ((inf2!=sup) && x.ord(a)) {
                       inf2 = inf2.succ();
                       x = inf2.val();
                       met = met || (sup1==inf2);
                   }
                   T y = sup1.val();
                   while ((sup1!=inf) && a.ord(y)) {
                       sup1 = sup1.pred();
                       y = sup1.val();
                       met = met || (sup1==inf2);
                   }
                   if (!met) {
                       try {sup1.swap(inf2);}
                       catch (Exception e) {
                           System.out.println("errore di metodo in divisione:");
                           System.out.println("--invocato swap con null reference");
                           System.out.println("--swap non eseguito");
                       }
                   }
               }
               if (inf2 == sup1) sup1 = sup1.pred();
               return new pair(sup1,inf2);
           }
           void QSort(myMutSeq<T> cInf, myMutSeq<T> cSup){
               if (cInf == cSup) return;
               {
```

```
                    T a;
                    try {a = separatore(cSup);}
                    catch (Exception e) {
                        System.out.println("errore di programma in QSort:");
                        System.out.println("--invocato separatore con null reference");
                        System.out.println("--terminata esecuzine di QSort");
                        return;
                    }
                    pair sup1inf2 = divisione(cInf,cSup,a);
                    QSort(cInf,sup1inf2.left);
                    QSort(sup1inf2.right,cSup);
                }
            }
        }
        new sortStuff().QSort(first(),last());
    }
}
```